COS420: Program 3

By Abigail Pitcairn

February 12, 2025

For this comparison, I used ChatGPT-4-turbo, which is the default model when you access chatgpt.com. The most obvious differences were in the **time to complete the assignment** and the **design choices for the output**.

Completing the assignment with only human efforts took around **an hour**, but note as well that I have already completed this assignment for an independent study, therefore I was already familiar and able to go quickly. Using AI tools, it took only about **twenty minutes** and very little actual effort. The main time constraints using AI tools was just waiting for the AI to generate the code and copy/pasting that output into Eclipse for testing. ChatGPT required only two follow-up prompts after being given the Bulldog program directions; one follow-up to add documentation, and another follow-up to put everything all together because when ChatGPT added documentation, it removed WimpPlayer.

The second biggest difference that I saw in the programs was with the formatting of output. I chose a menu that had users put in the number of players, then player by player, input the player type and name. ChatGPT did a very similar menu on the first return, but after the aforementioned follow-up prompts for documentation and adding back WimpPlayer, **ChatGPT returned a different menu** that gave numbers for each player type, so when users would select the player type they would input '1' for HumanPlayer, '2' for UniquePlayer, and so on.

To further elaborate on the quality of the code, both mine and ChatGPT's seemed correct on their final iterations, but **ChatGPT's documentation was far worse.** For example, even when explicitly asked to add documentation, the entirety of the documentation for FifteenPlayer was just this single comment at the top of the class:

```
/**
* A player that stops rolling once their turn score reaches 15.
*/
```

My own documentation looked like this:

```
/*********************************************************/
/* Abigail Pitcairn                                      */
/* COS 420, Spring 2025                                  */
/* FifteenPlayer: subclass of Player                     */
/* Each turn, roll until reaching 15 or rolling a 6      */
/*********************************************************/
public class FifteenPlayer extends Player {
  /**Constructor
   * @param name - name of the player
   */
  public FifteenPlayer(String name) {…}

  /**Execute FifteenPlayer's turn.
```

```
     * FifteenPlayer will roll dice until their score for the current turn
     * is greater than or equal to 15, or if a six is rolled the score is 0.
     * @return FifteenPlayer's final score for the current turn.
     */
    @Override
    public int play() {...}
}
```

And the Player.java class, which was provided in full to ChatGPT for reference, had very detailed documentation, as we know. I do believe that with further prompting, the AI's documentation could have been better, but it was becoming easier for me to do it myself, so for the sake of comparison I kept ChatGPT's subpar comments as the final product.

The main takeaway I have from this assignment is that ChatGPT is a great tool for generating a starting point, but it is far from delivering a perfect final product. With each further prompt you give it, you risk it changing something that you didn't want changed, and what it changes isn't always noticeable. It's fully possible to run in circles telling it to fix problem A, and in fixing problem A, it causes problem B; you prompt it to fix B and it breaks A again, or causes a new problem, and so on.

Additionally, these problems seem completely random: there is no clear way to control whether or not they appear, and it ends up with the programmer (me) manually intervening to combine two of ChatGPT's partially-correct answers. If you know what you're doing, ChatGPT is a great shortcut. If you don't know what you're doing, ChatGPT will gaslight you into thinking it's given you the right answer, or you'll run in "prompt circles" saying: "fix A", "now fix B," "now fix C", "now fix A again"...

To support my point, I will further elaborate on the follow-up prompts I used on the Bulldog program with AI, using simplified language so that the flow is easier to understand:

1. "Add documentation"
    a. removed the WimpPlayer code
    b. removed WimpPlayer from the menu
    c. reformatted the menu from "enter player type (Human, Fifteen,...)" to "enter 1 for Human; 2 for Fifteen;..."
2. "Add WimpPlayer"
    a. added WimpPlayer as subclass to Player class
    b. did not return any code for program 6, although the previous iteration of program 6 did not include WimpPlayer; WimpPlayer had to be manually added to the program 6 options.

I decided to stop prompting after these two prompts to avoid spending too much time on the AI tool, as that defeats the purpose of the AI tool if I can do it faster by myself.

If I were to do this assignment again, I would include more details in the original prompt to attempt to get the best result on the first AI output. For example, "add thorough JavaDoc to all classes, constructors and methods" or "add all five subclasses of Player to the menu". However, these points can really only be made in hindsight; you don't know where the AI will lack on a particular task until you have it do said task at least once. Overall, the AI did save me time and it did a good job. This is likely because this program is small, simple and has very few dependencies, and because there was already lots of information in the Bulldog assignment description that I could quickly copy and paste as a prompt.